1st Asian-Pacific Summer School on Formal Methods
Course 12: Deductive verification of C programs with Frama-C and Jessie

Virgile Prevosto

CEA List

August 30, 2009

outline

- Jessie Usage
- Function Contracts
- Advanced Specification
  - Example 1: Searching
  - Example 2: Sorting

Jessie Usage

Function Contracts

Advanced Specification

Example 1: Searching

Example 2: Sorting

frama C
Software Analyzers

What is Jessie?

- Hoare-logic based plugin, developed at INRIA Saclay.
- Input: a program and a specification
- Jessie generates verification conditions
- Use of Automated Theorem Provers to discharge the VCs
- If all VCs are proved, the program is correct with respect to the specification
- Otherwise: need to investigate why the proof fails
  - Fix bug in the code
  - Adds additional annotations to help ATP
  - Interactive Proof (Coq)

What is Jessie?

- Hoare-logic based plugin, developed at INRIA Saclay.
- Input: a program and a specification
- Jessie generates verification conditions
- Use of Automated Theorem Provers to discharge the VCs
- If all VCs are proved, the program is correct with respect to the specification
- Otherwise: need to investigate why the proof fails
  - Fix bug in the code
  - Adds additional annotations to help ATP
  - Interactive Proof (Coq)

What is Jessie?

- Hoare-logic based plugin, developed at INRIA Saclay.
- Input: a program and a specification
- Jessie generates verification conditions
- Use of Automated Theorem Provers to discharge the VCs
- If all VCs are proved, the program is correct with respect to the specification
- Otherwise: need to investigate why the proof fails
  - Fix bug in the code
  - Adds additional annotations to help ATP
  - Interactive Proof (Coq)

What is Jessie?

- Hoare-logic based plugin, developed at INRIA Saclay.
- Input: a program and a specification
- Jessie generates verification conditions
- Use of Automated Theorem Provers to discharge the VCs
- If all VCs are proved, the program is correct with respect to the specification
- Otherwise: need to investigate why the proof fails
  - Fix bug in the code
  - Adds additional annotations to help ATP
  - Interactive Proof (Coq)

What is Jessie?

- Hoare-logic based plugin, developed at INRIA Saclay.
- Input: a program and a specification
- Jessie generates verification conditions
- Use of Automated Theorem Provers to discharge the VCs
- If all VCs are proved, the program is correct with respect to the specification
- Otherwise: need to investigate why the proof fails
  - Fix bug in the code
  - Adds additional annotations to help ATP
  - Interactive Proof (Coq)

What is Jessie?

- Hoare-logic based plugin, developed at INRIA Saclay.
- Input: a program and a specification
- Jessie generates verification conditions
- Use of Automated Theorem Provers to discharge the VCs
- If all VCs are proved, the program is correct with respect to the specification
- Otherwise: need to investigate why the proof fails
  - Fix bug in the code
  - Adds additional annotations to help ATP
  - Interactive Proof (Coq)

What is Jessie?

- Hoare-logic based plugin, developed at INRIA Saclay.
- Input: a program and a specification
- Jessie generates verification conditions
- Use of Automated Theorem Provers to discharge the VCs
- If all VCs are proved, the program is correct with respect to the specification
- Otherwise: need to investigate why the proof fails
  - Fix bug in the code
  - Adds additional annotations to help ATP
  - Interactive Proof (Coq)

What is Jessie?

- Hoare-logic based plugin, developed at INRIA Saclay.
- Input: a program and a specification
- Jessie generates verification conditions
- Use of Automated Theorem Provers to discharge the VCs
- If all VCs are proved, the program is correct with respect to the specification
- Otherwise: need to investigate why the proof fails
  - Fix bug in the code
  - Adds additional annotations to help ATP
  - Interactive Proof (Coq)

What is Jessie?

- Hoare-logic based plugin, developed at INRIA Saclay.
- Input: a program and a specification
- Jessie generates verification conditions
- Use of Automated Theorem Provers to discharge the VCs
- If all VCs are proved, the program is correct with respect to the specification
- Otherwise: need to investigate why the proof fails
  - Fix bug in the code
  - Adds additional annotations to help ATP
  - Interactive Proof (Coq)

What is Jessie Useful for?

Usage

- Proof of functional properties of the program
- Modular verification (function per function)

Limitations

- Cast between pointers and integers
- Limited support for union type
- Aliasing requires some care

Software Analyzers

What is Jessie Useful for?

Usage

- Proof of functional properties of the program
- Modular verification (function per function)

Limitations

- Cast between pointers and integers
- Limited support for union type
- Aliasing requires some care

What is Jessie Useful for?

Usage

- Proof of functional properties of the program
- Modular verification (function per function)

Limitations

- Cast between pointers and integers
- Limited support for union type
- Aliasing requires some care

What is Jessie Useful for?

Usage

- Proof of functional properties of the program
- Modular verification (function per function)

Limitations

- Cast between pointers and integers
- Limited support for union type
- Aliasing requires some care

What is Jessie Useful for?

Usage

- Proof of functional properties of the program
- Modular verification (function per function)

Limitations

- Cast between pointers and integers
- Limited support for union type
- Aliasing requires some care

What is Jessie Useful for?

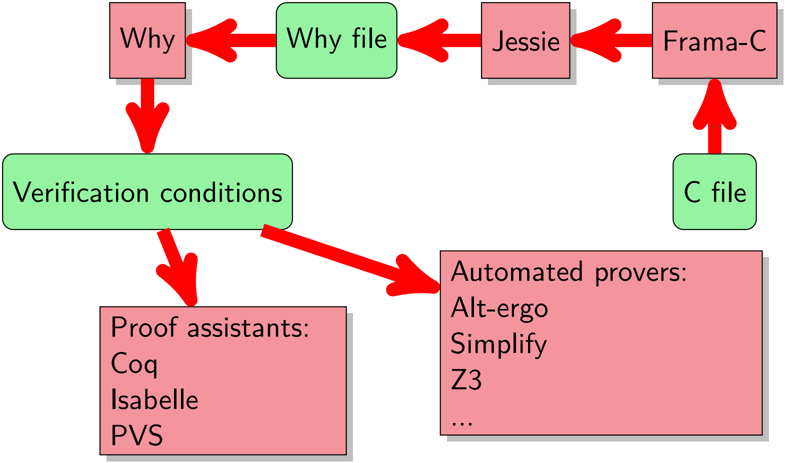
Usage

- Proof of functional properties of the program
- Modular verification (function per function)

Limitations

- Cast between pointers and integers
- Limited support for union type
- Aliasing requires some care

From Frama-C to Theorem Provers



A first example

Check safety of a function

- Pointer accesses
- Arithmetic overflow
- Division

```
unsigned int M;

void mean(unsigned int* p, unsigned int* q) {
  M = (*p + *q) / 2;
}
```

Safety of a program is important, but this is not sufficient: We also want it to do “the right thing”...

But in order for jessie to verify that, we need to explain it what “the right thing” is, and to explain it formally

This is the purpose of ACSL, ANSI/ISO C Specification Language.

- Behavioral specification language à la JML and Eiffel
- Function contracts
- Logic models
- Independent from any plug-in

Safety of a program is important, but this is not sufficient: We also want it to do "the right thing"...

But in order for jessie to verify that, we need to explain it what "the right thing" is, and to explain it formally

This is the purpose of ACSL, ANSI/ISO C Specification Language.

- Behavioral specification language à la JML and Eiffel
- Function contracts
- Logic models
- Independent from any plug-in

ACSL

Safety of a program is important, but this is not sufficient: We also want it to do “the right thing”...

But in order for jessie to verify that, we need to explain it what “the right thing” is, and to explain it formally

This is the purpose of ACSL, ANSI/ISO C Specification Language.

- Behavioral specification language à la JML and Eiffel
- Function contracts
- Logic models
- Independent from any plug-in

ACSL

Safety of a program is important, but this is not sufficient: We also want it to do "the right thing"...
But in order for jessie to verify that, we need to explain it what "the right thing" is, and to explain it formally
This is the purpose of ACSL, ANSI/ISO C Specification Language.

- Behavioral specification language à la JML and Eiffel
- Function contracts
- Logic models
- Independent from any plug-in

ACSL

Safety of a program is important, but this is not sufficient: We also want it to do “the right thing”...
But in order for jessie to verify that, we need to explain it what “the right thing” is, and to explain it formally
This is the purpose of ACSL, ANSI/ISO C Specification Language.

- Behavioral specification language à la JML and Eiffel
- Function contracts
- Logic models
- Independent from any plug-in

ACSL

Safety of a program is important, but this is not sufficient: We also want it to do “the right thing”...
But in order for jessie to verify that, we need to explain it what “the right thing” is, and to explain it formally
This is the purpose of ACSL, ANSI/ISO C Specification Language.

- Behavioral specification language à la JML and Eiffel
- Function contracts
- Logic models
- Independent from any plug-in

ACSL

Safety of a program is important, but this is not sufficient: We also want it to do "the right thing"...
But in order for jessie to verify that, we need to explain it what "the right thing" is, and to explain it formally
This is the purpose of ACSL, ANSI/ISO C Specification Language.

- Behavioral specification language à la JML and Eiffel
- Function contracts
- Logic models
- Independent from any plug-in

Safety of a program is important, but this is not sufficient: We also want it to do “the right thing”...

But in order for jessie to verify that, we need to explain it what “the right thing” is, and to explain it formally

This is the purpose of ACSL, ANSI/ISO C Specification Language.

- Behavioral specification language à la JML and Eiffel
- Function contracts
- Logic models
- Independent from any plug-in

Behavior of a function

- Functional specification
- Pre-conditions (requires)
- Post-conditions (ensures)

Example

```
unsigned int M;
/*@
  requires \valid(p) ∧ \valid(q);
  ensures M ≡ (*p + *q) / 2;
*/
void mean(unsigned int* p, unsigned int* q) {
  if (*p ≥ *q) { M = (*p − *q) / 2 + *q; }
  else { M = (*q − *p) / 2 + *p; }
}
```

Behavior of a function

- Functional specification
- Pre-conditions (requires)
- Post-conditions (ensures)

Example

```
unsigned int M;
/*@
  requires \valid(p) ∧ \valid(q);
  ensures M ≡ (*p + *q) / 2;
*/
void mean(unsigned int* p, unsigned int* q) {
  if (*p ≥ *q) { M = (*p − *q) / 2 + *q; }
  else { M = (*q − *p) / 2 + *p; }
}
```

Behavior of a function

- Functional specification
- Pre-conditions (`requires`)
- Post-conditions (`ensures`)

Example

```
unsigned int M;
/*@
  requires \valid(p) ∧ \valid(q);
  ensures M ≡ (*p + *q) / 2;
*/
void mean(unsigned int* p, unsigned int* q) {
  if (*p ≥ *q) { M = (*p − *q) / 2 + *q; }
  else { M = (*q − *p) / 2 + *p; }
}
```

Side effects

The specification:

```
/*@
  requires \valid(p) ∧ \valid(q);
  ensures M ≡ (*p + *q) / 2;
  assigns M;
*/
void mean(unsigned int* p, unsigned int* q);
```

Jessie Usage

Function Contracts

Advanced Specification

- Example 1: Searching
- Example 2: Sorting

Frama-C Software Analyzers

A concrete example

Informal spec

- Input: a sorted array and its length, an element to search.
- Output: index of the element or −1 if not found

Implementation

```
int find_array(int* arr, int length, int query) {
  int low = 0;
  int high = length – 1;
  while (low ≤ high) {
    int mean = low + (high –low) / 2;
    if (arr[mean] ≡ query) return mean;
    if (arr[mean] < query) low = mean + 1;
    else high = mean – 1;
  }
  return –1;
}
```

An example

Informal specification

- Input: an array and its length
- Output: the array is sorted in ascending order

```
int index_min(int* a, int low, int high);

void swap(int* arr, int i, int j);

void min_sort(int* arr, int length) {
  for(int i = 0; i < length; i++) {
    int min = index_min(arr,i,length);
    swap(arr,i,min);
  }
}
```